

Adaptive Precision Arithmetic for Reducing Memory and Increasing Computation*

P. D. Hovland, T. Munson, and S. M. Wild

*Mathematics and Computer Science Division
Argonne National Laboratory, Argonne, IL 60439*

May 2013

Constraints on both memory size and the power required to move data imply that extreme-scale computations will need to balance their memory footprint and computational requirements. Adaptive precision arithmetic offers the potential to obtain better information using less memory and more floating-point operations. These techniques should become a standard tool for computational scientists, and be applied during all parts of the computation, from nonlinear solvers that require only greater precision as they approach a solution, to hierarchical methods that may not need high precision at all levels. Reducing the memory footprint using these techniques may improve resilience by reducing the probability of bit flips. Effective use of adaptive precision arithmetic requires mathematical analysis of the computations and solvers to determine their applicability, flexible libraries that allow one to seamlessly change precision at run time, and storage schemes that can be accelerated in hardware.

Adaptive precision arithmetic can play a fundamental role in reducing memory usage for large-scale applications. Global requirements on the precision of the computations during each iteration of Newton's method, for example, could be derived from the Dembo, Eisenstat, and Steihaug conditions for solving nonlinear systems of equations, which provide guidance as to how accurately the linear systems of equations need to be solved to retain a fast convergence rate. Far from a solution, the requirements are loose and can be performed with lower-precision arithmetic; near a solution, tighter tolerances necessitate higher-precision arithmetic. Once in the domain of fast (quadratic) convergence, we require only a small number of iterations with higher-precision arithmetic to compute an accurate solution. Techniques such as iterative refinement, where one stores the matrix in lower precision but performs matrix-vector products in higher precision, could also be applied to improve the accuracy of the solution to a linear system of equations at a low computational cost.

Multigrid methods for solving partial differential equations and other hierarchical methods could also benefit from adaptive precision arithmetic. In multigrid, for example, one uses a sequence of meshes and traverses these meshes from fine to coarse and back by using interpolation and restriction operations. Such methods are nearly optimal for solving many elliptic partial differential equations. By analyzing the theory of multigrid methods, one could exploit the hierarchical structure to determine the precision required at each level in the mesh hierarchy. By making the optimal precision choices at each level, a method could be obtained that uses a reduced memory footprint yet retains the optimal complexity at a reduced computational cost per iteration. Moreover, the technique of Giles and Pierce, which corrects discretization error in a function value by using derivatives, may be adapted to compensate or correct for representation errors resulting from the use of lower-precision arithmetic. Other hierarchical methods, such as the branch-and-bound method for solving discrete optimization problems, could also use adaptive precision arithmetic. In this case the branch-and-bound method recursively subdivides the domain and solves an optimization problem on each subdomain. Most of these optimizations could be performed in lower precision; only when nodes are pruned is a higher precision required.

*This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

Given the precision needed for a particular computation, in many cases it may be sufficient to compute the required functions and derivatives using lower precision. In these cases, one may adopt different precision levels for the different terms, from double-precision function values, to single-precision gradients, to half-precision Hessians. Analysis tools, such as those developed for automatic differentiation and estimating computational noise, could identify blocks in the code for which higher precision would lead to improved precision in function evaluations. Based on this identification, one could restructure the computation of a function so that the least-precision arithmetic is used in each block to obtain the required precision in the overall function evaluation. Similar ideas could be applied for gradient and Hessian evaluations. The asymptotic convergence properties of Newton’s method depend heavily on the quality of these derivative approximations, and hence need to be more accurate as a solution is approached. When analytic derivatives are computed using a lower-precision representation, however, the precision can be as high as the best one-sided finite-difference approximation using a higher-precision representation, because of the latter’s need to balance rounding and truncation errors. Moreover, user-provided and automatically generated codes for quantities derived from function values (such as derivatives) can be significantly less precise than the underlying function. Analysis of the computational graph (for example, as done by Kubota) could provide insight into reformulations of the derived code that yield both function and derived values to specified precision.

The energy and memory bandwidth cost of moving 64 bits may be better spent on moving two 32-bit or four 16-bit words than a single 64-bit word and increasing the computations performed with those words. By using two or more lower-precision words, one can compute not just an approximate value but also a rigorous or estimated bound on the error in that approximation. The error can be bounded rigorously by using interval arithmetic or the remainder differential algebra techniques of Berz. The error can be estimated statistically by computing two or more nearby points to estimate the noise or by using adjoint-based error estimates. Thus, these methods can obtain additional useful information using the same or a reduced memory footprint.

Local analysis within a finite-element method, for example, could also be used to compress the solution by applying adaptive-precision arithmetic. In particular, when the solution varies slowly over some regions, high-precision values are necessary only for some nodes in the mesh; the rest of the solution can be recovered by combining interpolation with a low-order correction. Efficiently determining the nodes needing high and low precision requires the development of appropriate algorithms for adapting the precision and logic to track the precision of each node.

By utilizing adaptive precision arithmetic in concert with interval analysis, one can perform better calculations with rigorous error bounds using less memory, thus increasing the size of problem that can be solved on extreme-scale architectures and reducing the potential for bit flips. With the necessary development of software libraries and training in applied mathematics, these could become fundamental tools of the scientific community for developing codes at scale. While many low-level libraries exist that implement adaptive precision arithmetic and interval methods, many high-level libraries, such as linear algebra libraries, either do not support such methods or do not allow one to seamlessly change the precision at run time. This lack of high-level support is a significant barrier to adopting such techniques. Mixed-precision arithmetic could also be used to further control the available precision. For example, a single-precision number coupled with a half-precision remainder could effectively provide single-and-a-half precision arithmetic at a cost of extra operations. Developing simulations at the highest level of abstraction and automatically generating code is one possible route forward. Hardware acceleration could also alleviate some of the computation requirements imposed by these methods.